Learning SMT Enumeration

Mikoláš Janota[1], Jelle Piepenbrock[1,2] Bartosz Piotrowski[1,3],

[1]Czech Technical University
[2]Radboud University Nijmegen
[3]University of Warsaw

AITP
9 September 2021
Aussois

Background

How to refute the following formula?

$$\forall x\, f(x) > x \land \forall y\, f(y) < 0$$

Background

How to refute the following formula?

$$\forall x\, f(x) > x \land \forall y\, f(y) < 0$$

$$x \mapsto 0$$

$$y \mapsto 0$$

Background

How to refute the following formula?

$$\forall x\, f(x) > x \wedge \forall y\, f(y) < 0$$

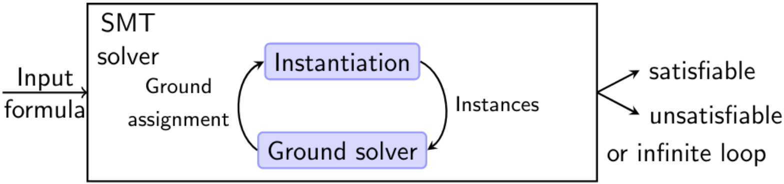
$$x \mapsto 0$$

$$y \mapsto 0$$

Then ground formula $f(0) > 0 \wedge f(0) < 0$ cannot be satisfied.

Background

Schematic of the SMT solver working with quantifiers:



Background

- Herbrand's theorem guarantees that for an unsatisfiable first-order logic formula, finitely many instantiations are sufficient to obtain an unsatisfiable ground part, and, these instantiations only need to use the Herbrand universe.
- A stronger variant of Herbrand's theorem that enables a more practical method for quantifier instantiation. It is sufficient to consider only the terms already within the ground part of the formula generated so far.
- This insight leads to the *enumerative instantiation* strategy.
- For a formula $G \wedge \forall x_1 \ldots x_n \, \phi$, with G ground, collect all ground terms $\mathcal{T}$ in G and strengthen G by an instantiation of ϕ by an n-tuple $t_1, \ldots, t_n$ with $t_i \in \mathcal{T}$; repeat the process until G becomes unsatisfiable or until resources are exhausted. The tuples are enumerated systematically to guarantee fairness.

Background

Let's consider the following conjunctive set of formulas within the logic of uninterpreted functions and linear integer arithmetic (UFLIA).

$$\{f(d) > f(d+2),\ c \leq 0 \vee \underbrace{\forall x\, f(x) < f(x+1)}_{q}\}$$

Background

Let's consider the following conjunctive set of formulas within the logic of uninterpreted functions and linear integer arithmetic (UFLIA).

$$\{f(d) > f(d+2),\ c \leq 0 \vee \underbrace{\forall x\, f(x) < f(x+1)}_{q}\}$$

Let's assume that the ground solver decided that $c \leq 0$ is false.

Background

Let's consider the following conjunctive set of formulas within the logic of uninterpreted functions and linear integer arithmetic (UFLIA).

$$\{f(d) > f(d+2),\ c \leq 0 \vee \underbrace{\forall x\, f(x) < f(x+1)}_{q}\}$$

Let's assume that the ground solver decided that $c \leq 0$ is false.

ground formula	**additional ground terms**
$\{c \leq 0 \vee q, f(d) > f(d+2)\}$	$\{d, d+2, c, 0, f(d), f(d+2)\}$

Background

Let's consider the following conjunctive set of formulas within the logic of uninterpreted functions and linear integer arithmetic (UFLIA).

$$\{f(d) > f(d+2),\ c \leq 0 \vee \underbrace{\forall x\, f(x) < f(x+1)}_{q}\}$$

Let's assume that the ground solver decided that $c \leq 0$ is false.

ground formula	**additional ground terms**
$\{c \leq 0 \vee q, f(d) > f(d+2)\}$	$\{d, d+2, c, 0, f(d), f(d+2)\}$
$\{q \Rightarrow f(d) < f(d+1)\}$	$\{d+1, f(d+1)\}$

Background

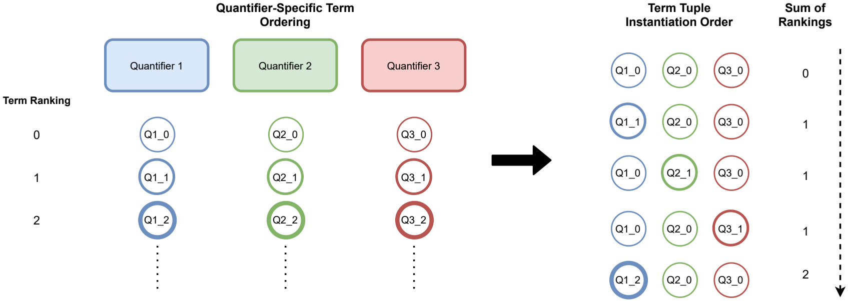
Let's consider the following conjunctive set of formulas within the logic of uninterpreted functions and linear integer arithmetic (UFLIA).

$$\{f(d) > f(d+2),\ c \leq 0 \vee \underbrace{\forall x\, f(x) < f(x+1)}_{q}\}$$

Let's assume that the ground solver decided that $c \leq 0$ is false.

ground formula	**additional ground terms**
$\{c \leq 0 \vee q, f(d) > f(d+2)\}$	$\{d, d+2, c, 0, f(d), f(d+2)\}$
$\{q \Rightarrow f(d) < f(d+1)\}$	$\{d+1, f(d+1)\}$
$\{q \Rightarrow f(d+1) < f(d+2)\}$	$\{d+2, f(d+2)\}$

Background

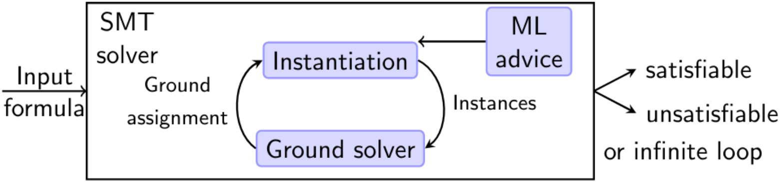


Quantifier-Specific Term Ordering
Quantifier 1
Quantifier 2
Quantifier 3
Term Ranking
0
1
2
Q1_0
Q1_1
Q1_2
Q2_0
Q2_1
Q2_2
Q3_0
Q3_1
Q3_2
Term Tuple Instantiation Order
Sum of Rankings
Q1_0 Q2_0 Q3_0 0
Q1_1 Q2_0 Q3_0 1
Q1_0 Q2_1 Q3_0 1
Q1_0 Q2_0 Q3_1 1
Q1_2 Q2_0 Q3_0 2

Can machine learning make SMT solvers more efficient in the context of quantifiers?

Machine learning guidance

Schematic of the SMT solver with machine learning guidance for quantifier instantiation.



Machine learning guidance

- Instead of ordering terms according to its age, we order them according to a scoring function $S\colon \mathcal{F} \to [0, 1]$.
- This function is parametrized as a machine-learning model – LightGBM.
- It takes as its argument the *features* $F(\phi, t)$ of a pair of a quantified sub-formula ϕ and the candidate term t which may be used for instantiation.
- It is trained on positive and negative examples:
  - (ϕ, t) is a positive if ϕ instantiated with t appeared in a proof
  - (ϕ, t) is a negative if instantiating ϕ with t was tried, but it did not appear in a proof.

Features

- *bag-of-words* (BOW) features:
  - we use *kinds* of symbols determined by CVC5 (like: *variable*, *skolem*, *not*, *and*, *plus*, *forall*, and many others)
  - we count how many times a given kind of symbol appeared
  - for example: $\mathrm{BOW}(\forall x\, 2 + x = \mathrm{skl}_1 + 3) = \{\mathit{forall} : 1, \mathit{variable}\text{: } \mathit{1}, \mathit{const} : 2, \mathit{skolem} : 1, \mathit{plus} : 2\}$
- additional features:
  - *varFrequency*
  - *age*
  - *phase*
  - *relevant*
  - *depth*
  - *tried*

Given an example (ϕ, t), its final feature representation is a vector

$$\mathrm{BOW}(\phi) + \mathrm{BOW}(t) + \text{additional features}$$

.

Data for evaluation

Three SMT-LIB benchmarks:

- UFLIA Sledgehammer
- UFNIA Sledgehammer
- UFLIA Boogie

Experimental setting

- One theorem may have multiple different proofs.
- One proof may result from multiple different proof-searches.
- This makes the notion of *positive / negative example* vague.

Experimental setting

Having a set of SMT problems, one can have two similar – but not equivalent – goals, which are equally important:

1. the *cumulative goal*: solve automatically as many of the problems as possible, running the ML-guided solver multiple times over them and improving it by training the ML model on data collected across the runs,
2. the *generalization goal*: use the available problems to train a single ML-guided solver which performs well on new, unseen problems.

Looping training and evaluation

Algorithm 1 Solving-training loop with training/testing split.

Require: training problems: P_{train}, testing problems: P_{test}, number of iterations: N, grid of hyper-parameters: H_{grid}

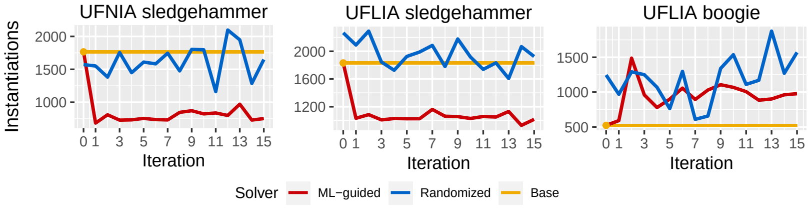
$M \leftarrow \{\}$
$D_{\text{train}} \leftarrow \{\}$
for $i \leftarrow 0$ *to* N **do**
    $L_{\text{train}} \leftarrow \textsc{Solve}(P_{\text{train}}, M)$
    $L_{\text{test}} \leftarrow \textsc{Solve}(P_{\text{test}}, M)$
    $D_{\text{train}} \leftarrow D_{\text{train}} \cup \textsc{ExtractTrainingExamples}(L_{\text{train}})$
    $H_{\text{best}} \leftarrow \textsc{GridSearch}(D_{\text{train}}, H_{\text{grid}})$
    $M \leftarrow \textsc{TrainModel}(D_{\text{train}}, H_{\text{best}})$

3 solvers compared in the experiments

1. **Base solver**: uses standard enumerative instantiation strategy
2. **Randomized solver**: like the base solver, but random 10% of terms are swapped
3. **ML-guided solver**: like the base solver, but terms are ordered according to ML-advice

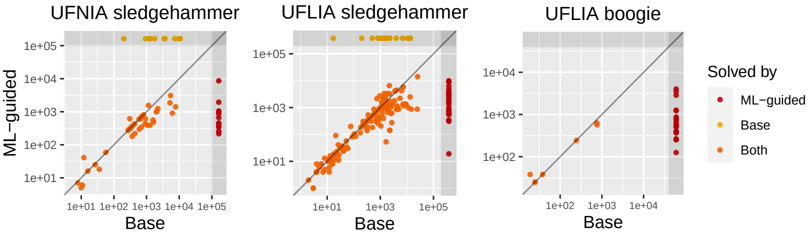
Results

Instantiations made by the solvers for testing problems across iterations of the evaluation:



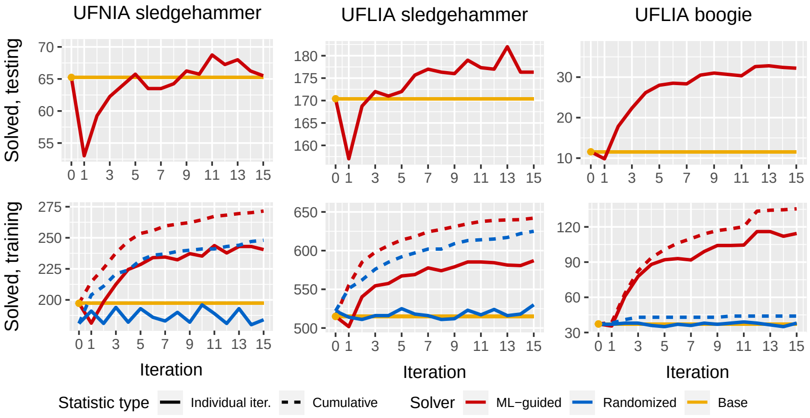
Results

Instantiations performed by the solvers
(each point refers to one testing problem):



Results

Numbers of problems solved in the looping evaluation for three benchmarks:



Future work

- Finding more clever way of dealing with tuples of variables.
- Designing more informative features.

Thank you!